

Data Plane Networking



Gaetano Borgione
Sr. Staff Engineer @ VMware





Gaetano Borgione
Senior Staff Engineer
Cloud Native Applications
VMWare

SDN Technologies @ PLUMgrid
Data Center Networking @ Cisco

Passionate Engineer with special interests on:
Networking Architecture
Engineering Leadership
Product Management
Customer Advocacy



+
...new Networking / Virtualization ideas !!!



Agenda

The background of the slide features a large white triangle on the left and a series of overlapping triangles on the right. These triangles are colored in various shades of green and blue, creating a modern, abstract geometric design.

Agenda

- BPF, eBPF, IOVisor project
- XDP
- Accelerating eBPF in Hardware

BPF, eBPF, IO Visor project

Introducing IO Visor Project

Evolution of Kernel
BPF & eBPF
(Berkeley Packet Filter)

Led by initial contributions
from PLUMgrid
(Upstreamed since Kernel 3.16)

Future of Linux Kernel IO
for software defined services

*“IO Visor will work closely with the Linux kernel community to **advance universal IO extensibility for Linux**. This collaboration is critically important as virtualization is putting more demands on flexibility, performance and security.*

*Open source software and collaborative development are the ingredients for addressing massive change in any industry. **IO Visor will provide the essential framework for this work on Linux virtualization and networking.**”*

Jim Zemlin, Executive Director, The Linux Foundation.

IO Visor Sponsoring Members



Special Thanks to Key Contributors:



1997



A little bit of history: BPF

- Introduced as Berkeley Packet Filters in kernel 2.1.75, in 1997
- BPF is now referred to as Classic BPF or cBPF
- Originally created as a way to analyze and filter network packets for network monitoring purposes
- BPF Goal: Accept packets you are interested in or discard them
- How: Userspace attaches a filter to a socket
- Example application: tcpdump/libpcap, wireshark, nmap, dhcp, arpd

A little bit of history: eBPF

2013



- e(xtended)BPF
- Initial proposal was in 2013, by [Alexei Starovoitov](#)* and up streamed since version 3.16
- Referred to as the universal in-kernel virtual machine
- Designed to give ability to create any in-kernel IO modules
- eBPF Goal: Improve and extend existing BPF infrastructure
- How: Programs in C and translated into eBPF instructions, loaded in kernel and executed. In-kernel compiler: x86, ARM64, s390, powerpc*, MIPS*
- Example Application: networking, tracing, security ...

[*https://lkml.org/lkml/2013/12/2/1066](https://lkml.org/lkml/2013/12/2/1066)

The eBPF Instruction Set

Instructions

- 10x 64bit registers
- 512B stack
- 1-8B load/store
- conditional jump
- arithmetic
- function call

Helper functions

- forward/clone/drop packet
- load/store packet data
- load/store packet metadata
- checksum (incremental)
- push/pop vlan
- access kernel mem (kprobes)

Data structures

- lookup/update/delete
 - in-kernel or from userspace
- hash, array, ...

www.iovisor.org



classic BPF	extended BPF
2 registers + stack 32-bit registers 4-byte load/store to stack 1-4 byte load from packet Conditional jump forward +, -, *, ... instructions	10 registers + stack 64-bit registers with 32-bit sub-registers 1-8 byte load/store to stack, maps, context Same + store to packet Conditional jump forward and backward Same + <u>signed_shift</u> + endian Call instruction <u>tail_call</u> map lookup/update/delete helpers packet rewrite, <u>csum</u>, <u>clone_redirect</u> <u>sk_buff</u> read/write

- Can build more complicated program
- Faster interpretation and JIT
- Support for calls to *approved* helper functions

BPF Kernel Hook Points

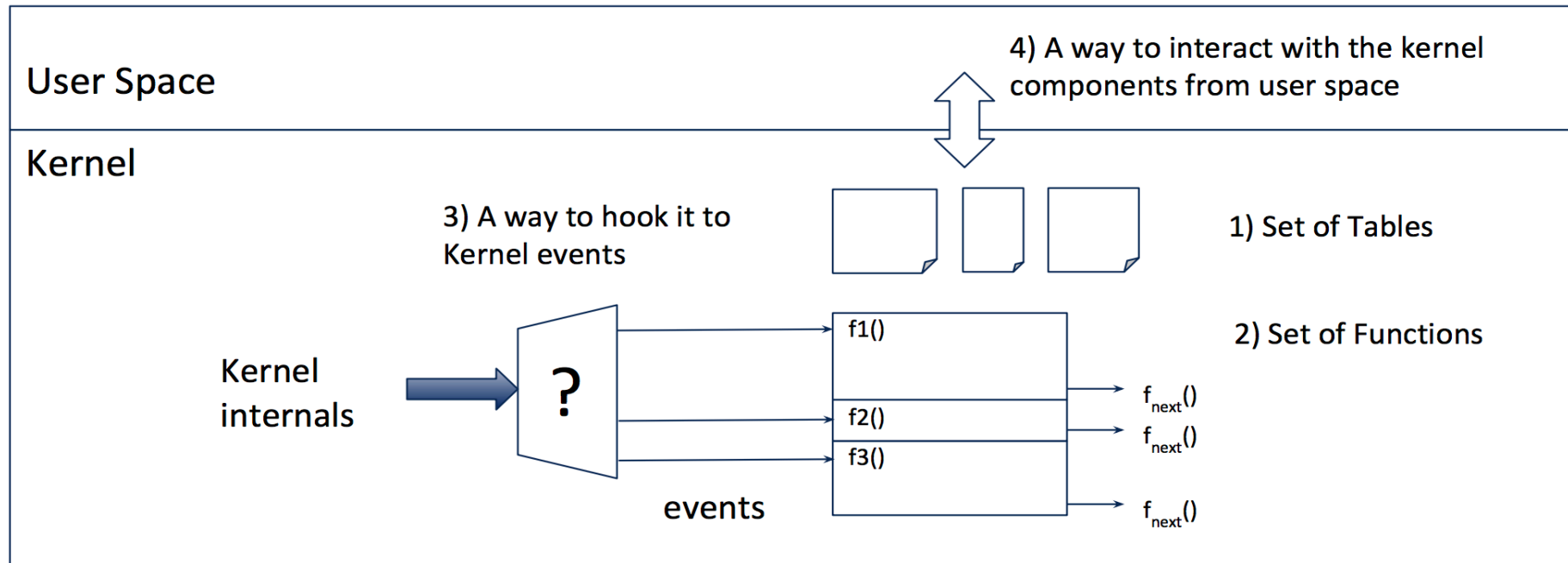
A program can be attached to:

- kprobes or uprobes
- socket filters (original tcpdump use case)
- tc filters or actions, either ingress or egress
- cgroups
- XDP

What are BPF Programs ?

In a very simplified way:

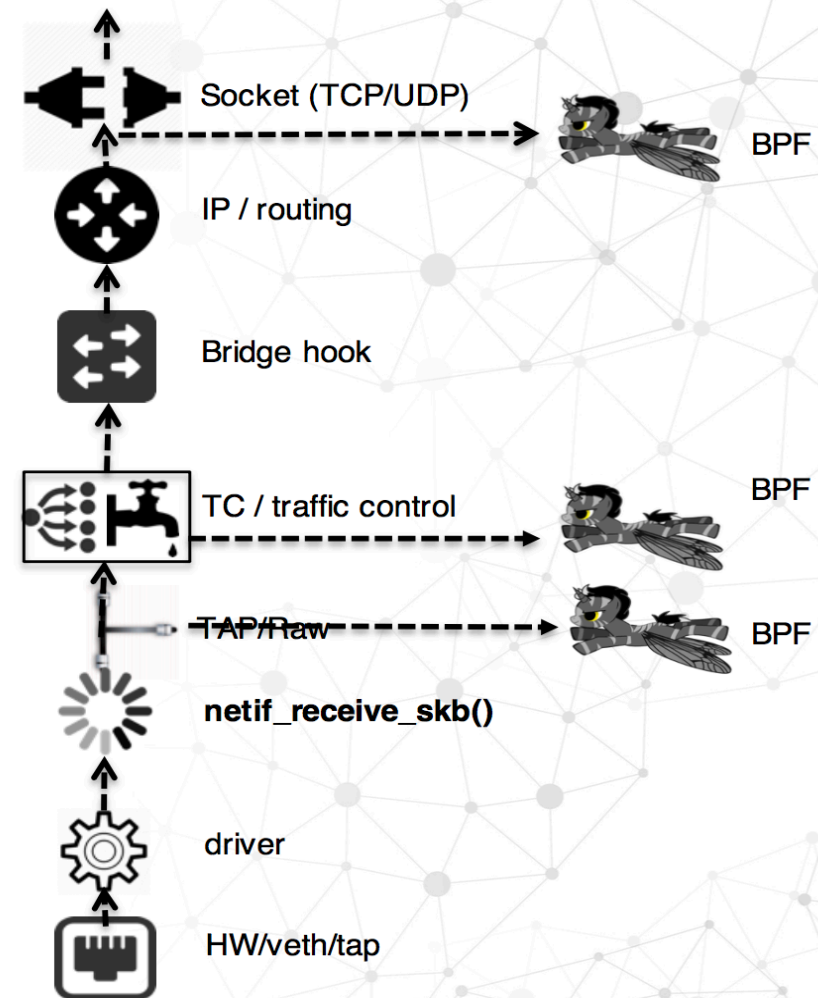
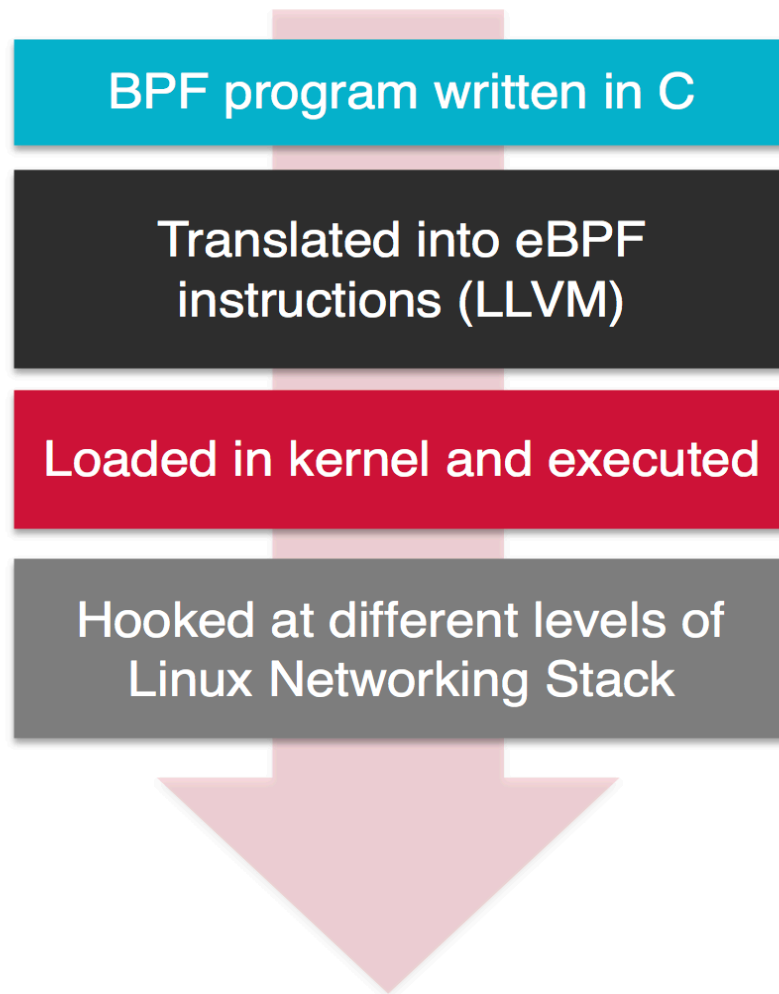
A safe, runtime way to extend Linux kernel capabilities
Functions, Maps, Attachment Points, Syscall



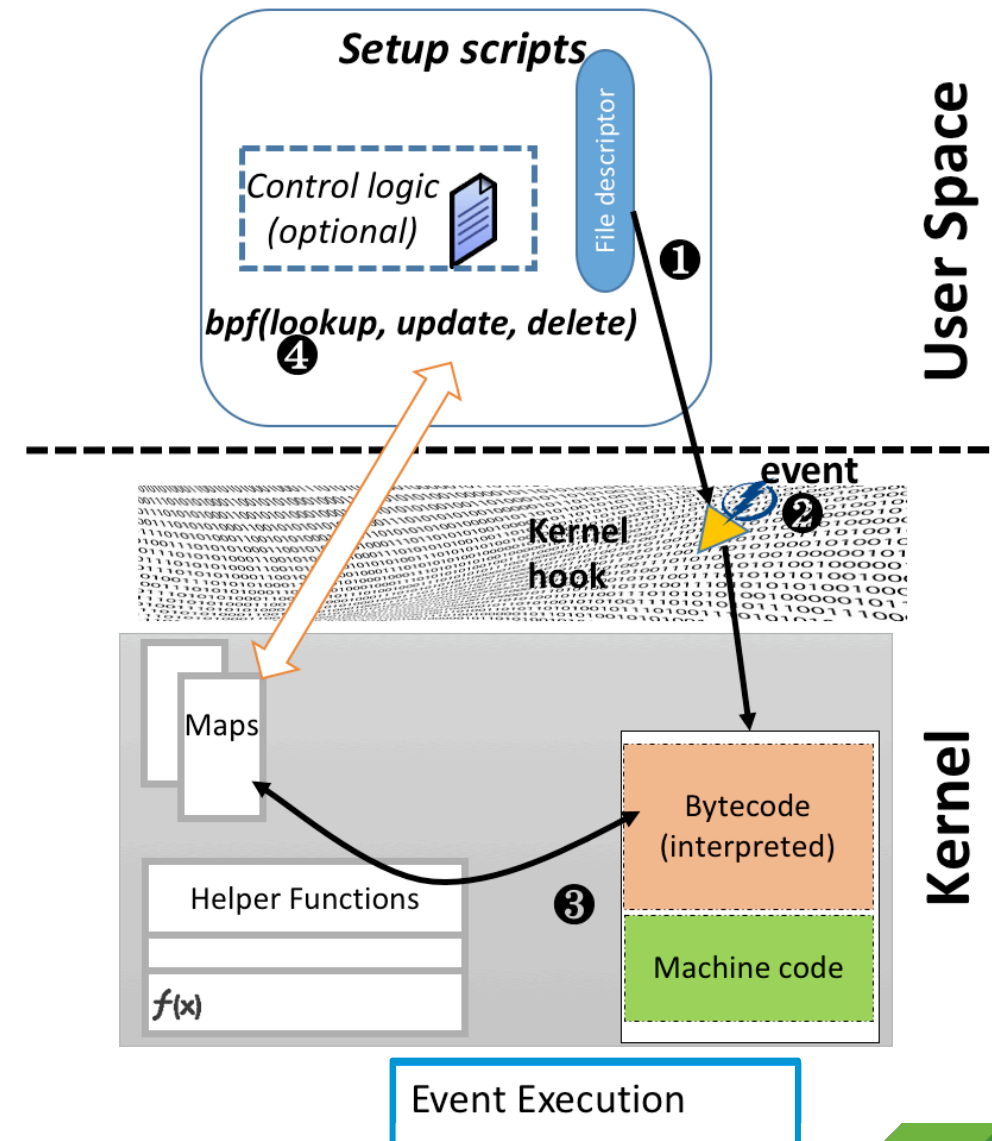
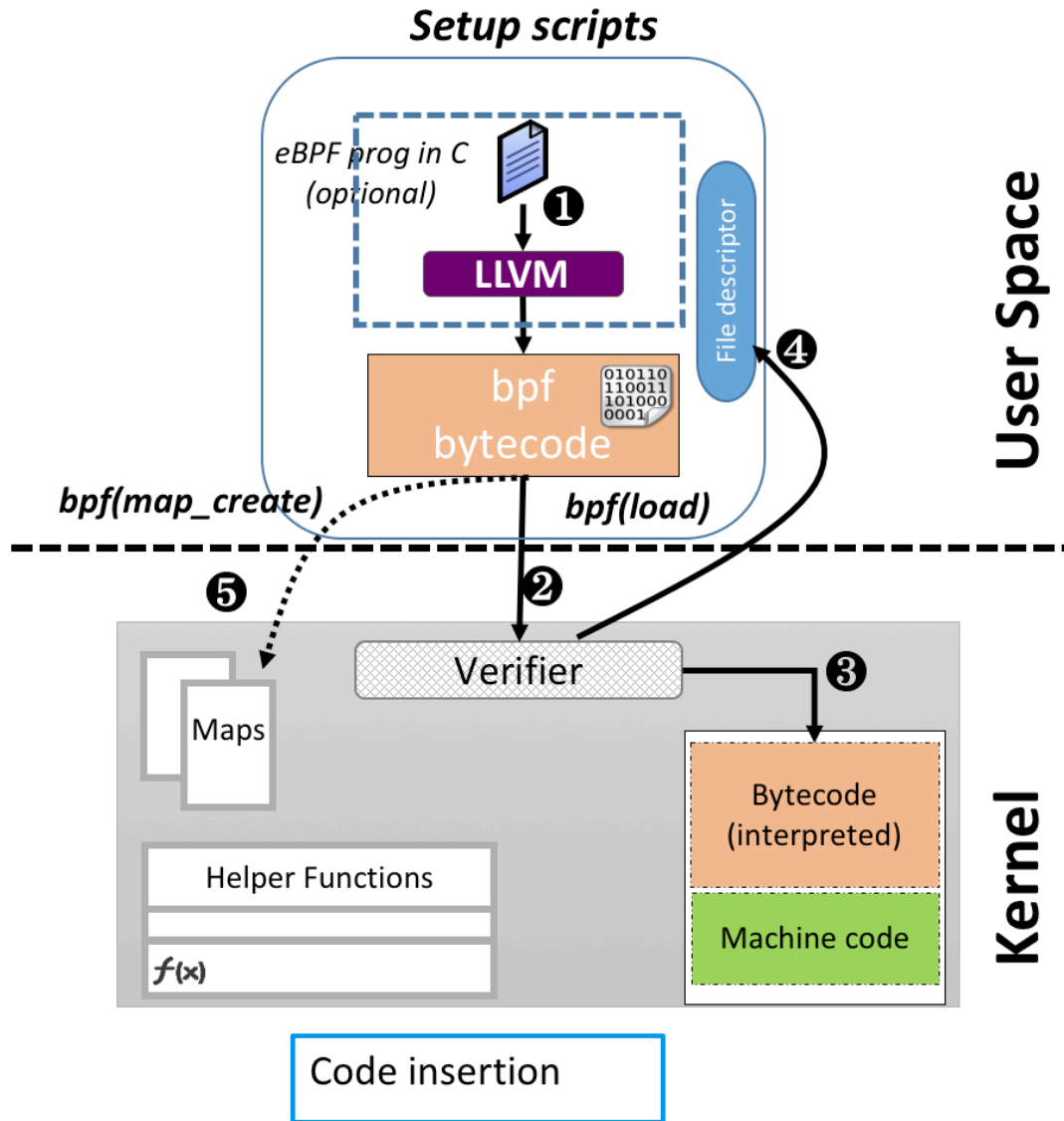
The concept of a BPF pseudo-machine

- Virtualize a machine instruction-set
- Write byte-code for this “fictional” machine
 - Verify this code is loop-free and optimized
- Interpret, in-kernel, for *any* real processor
- code is executed when an event occurs

eBPF: Loading New Modules



Visual Flow of Code Insertion and Execution



Verified and Kernel safety

- eBPF new architecture more complex
*required a **brand** new verifier*
- Provably confirms inserted program **does not**:
create loops, delays execution interminably, illegally dereference pointers
- Done statically, one-time, with an exhaustive search
some heuristics to improve verification time



eBPF to satisfy infrastructure app requirements...

- High performance access to data
- Reliability...it must never crash
- In-place upgrades
- Debug tools
- A programming language abstraction

...without restrictions

- No custom kernels
- No custom kernel modules
- No kernels with debug symbols
- No reboots !!!

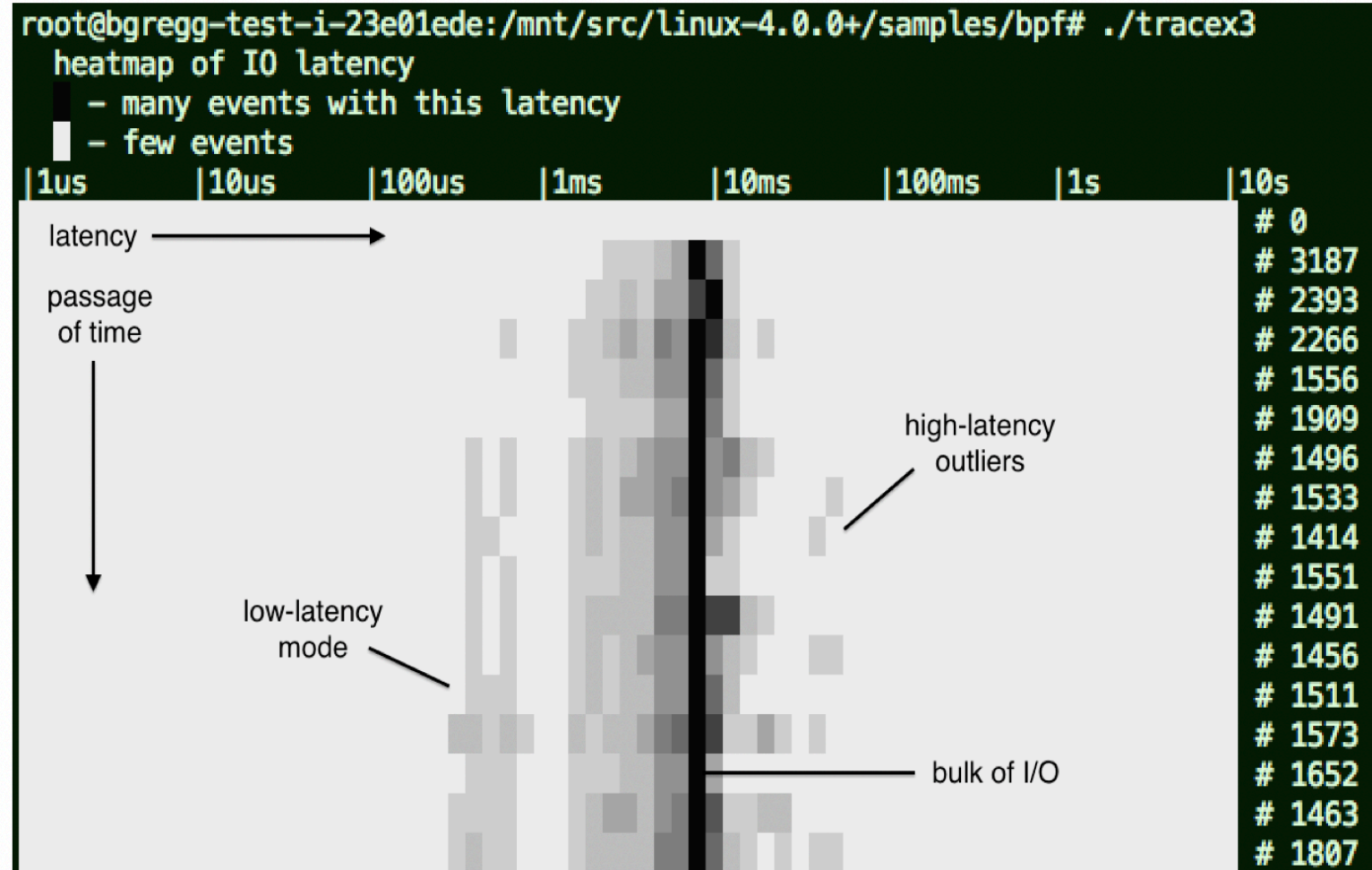
“Hello, World!” demo code

```
#!/usr/bin/python
import bcc
b = bcc.BPF(text="""
int kprobe__sys_clone(void *ctx) {
    bpf_trace_printk("Hello, World!\\n");
    return 0;
}
""")
b.trace_print()
```



Disk latency heat-maps

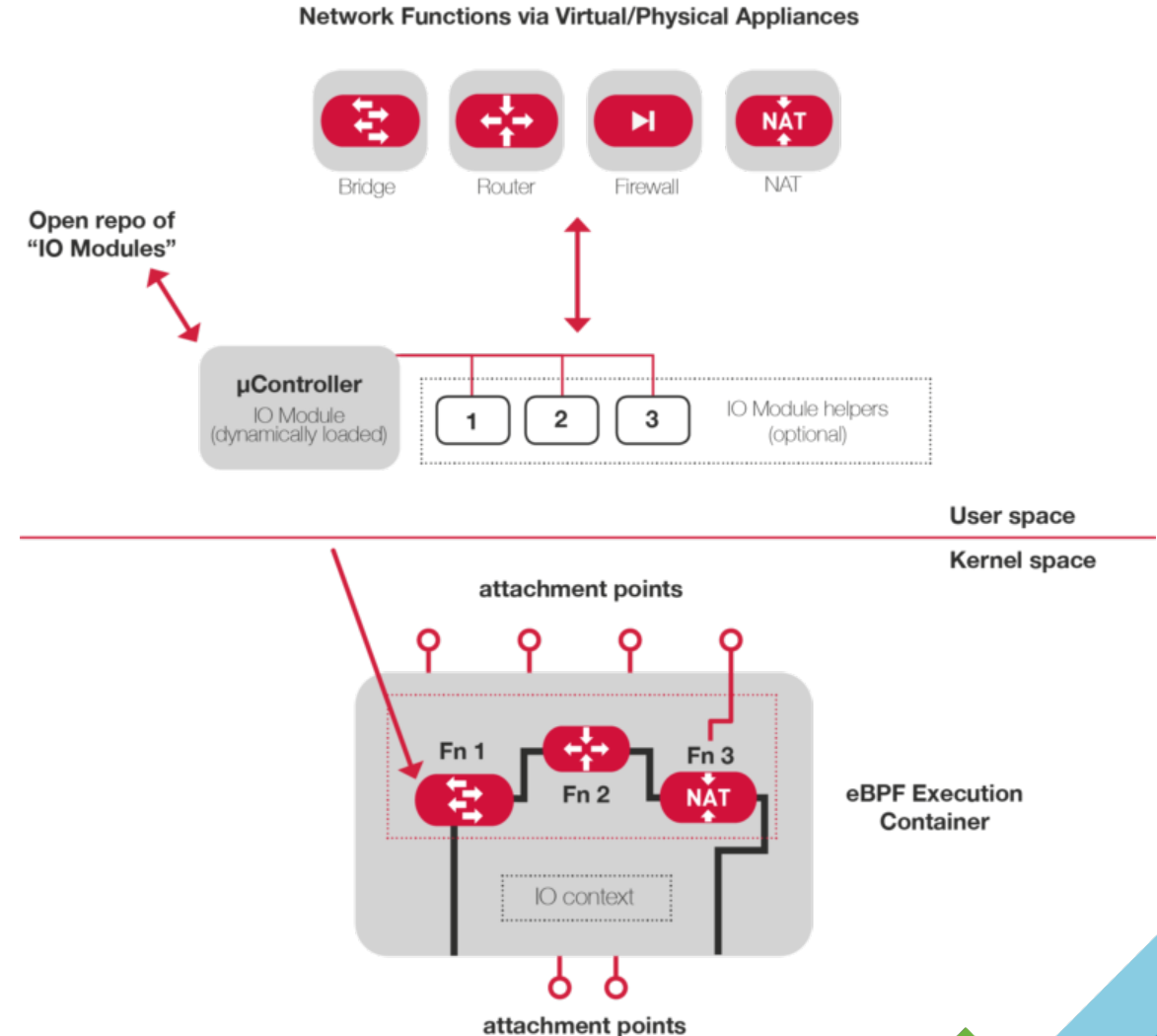
<https://github.com/torvalds/linux/tree/master/samples/bpf/>



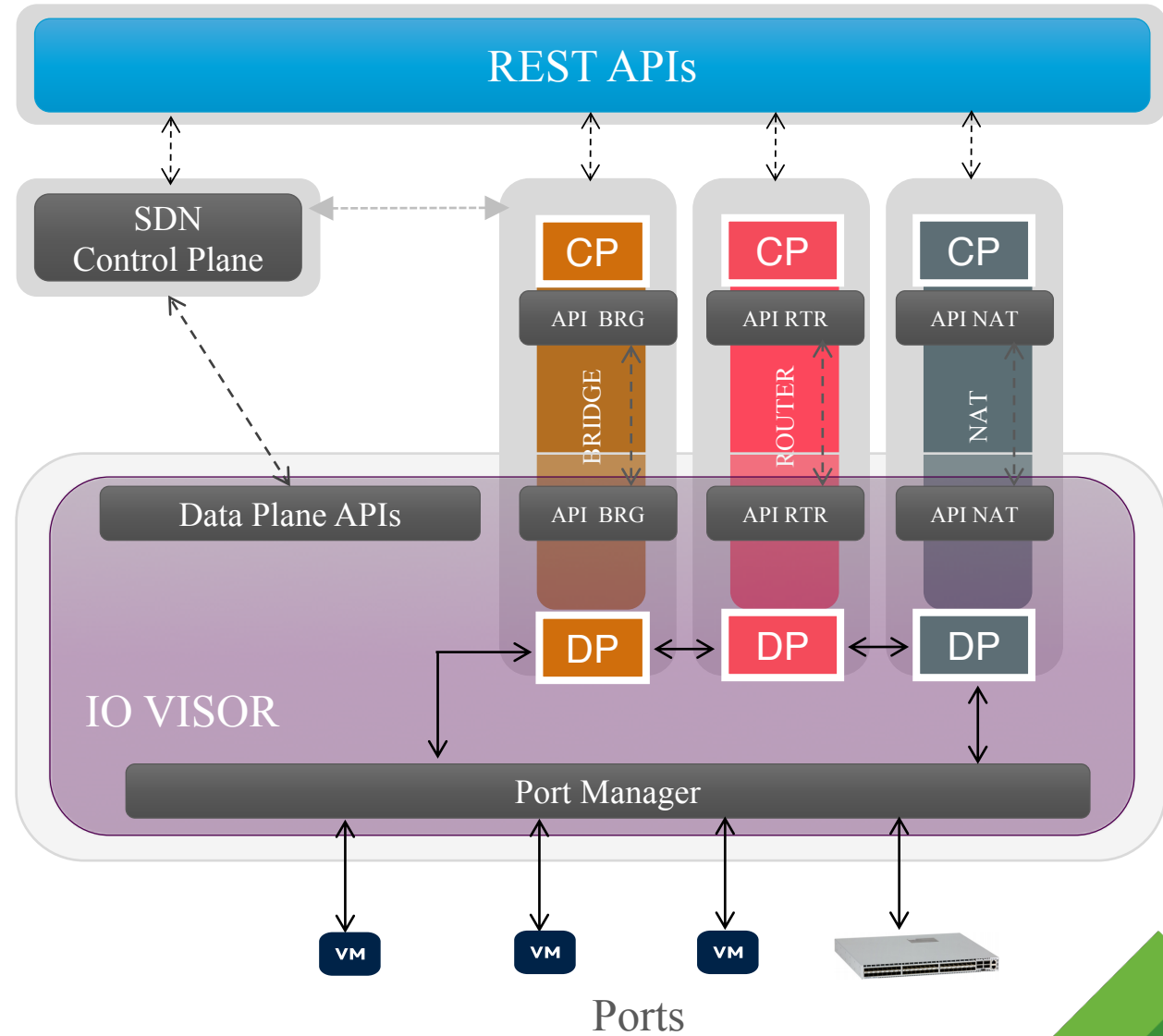
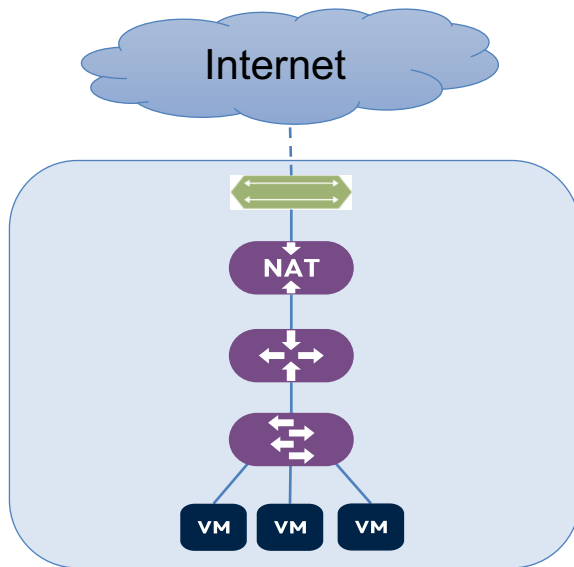
One map for time-stamp, other for latency

Networking as Use Case

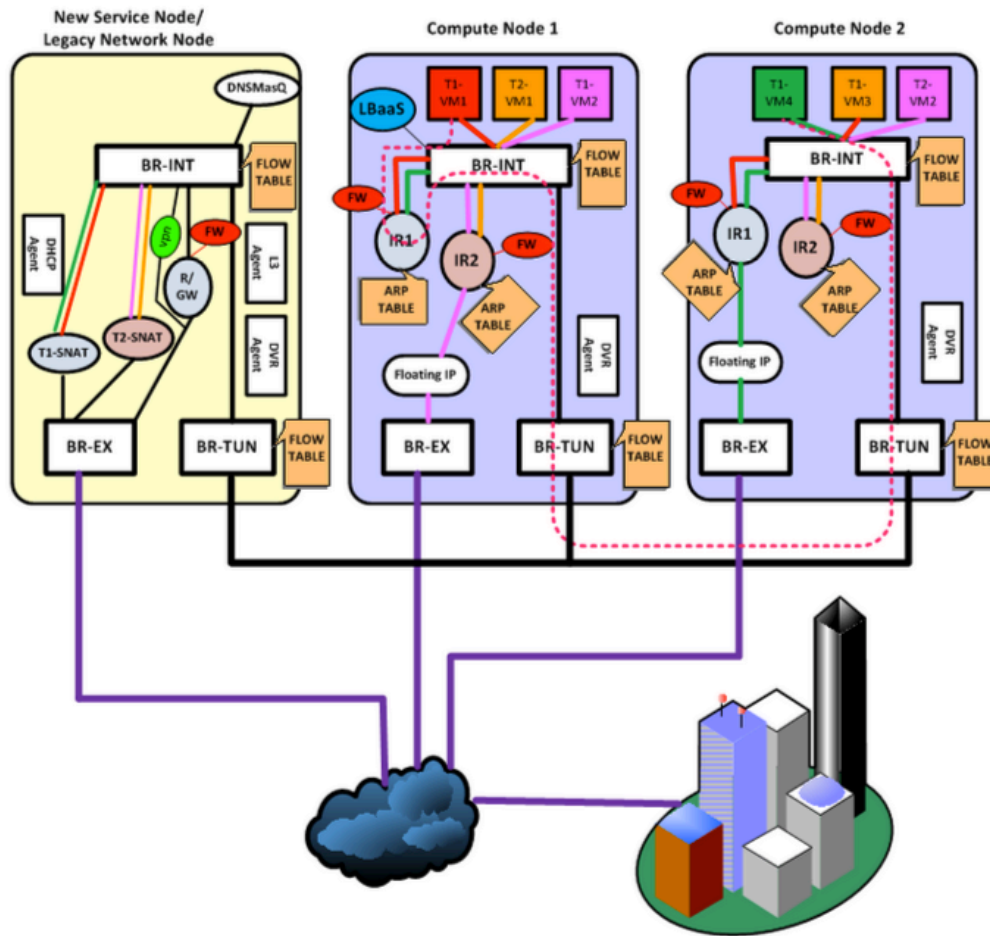
- IO Visor is used to build a fully distributed virtual network across multiple compute nodes
- All data plane components are inserted dynamically in the kernel
- No usage of virtual/physical appliances needed
- Example here
https://github.com/iovisor/bcc/tree/master/examples/distributed_bridge



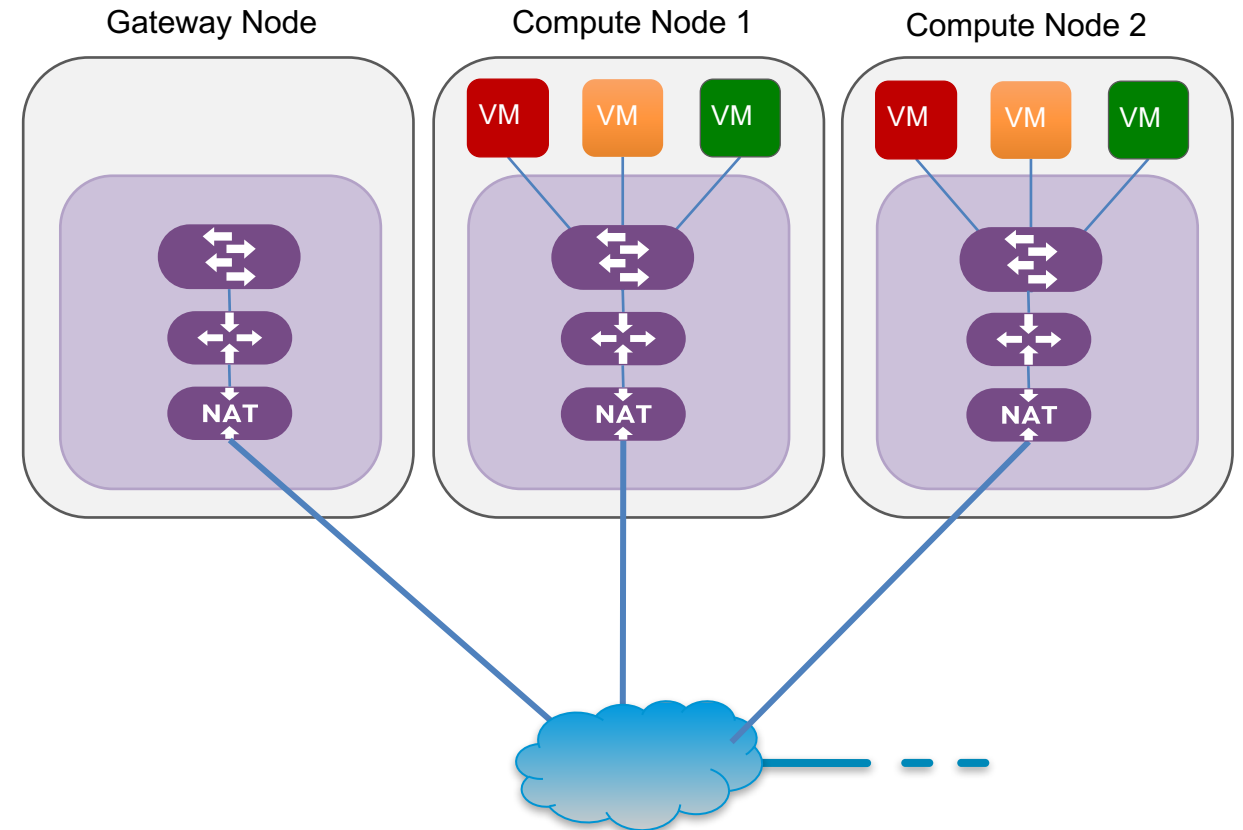
Networking as Use Case (cont.)



OVS and eBPF: different approaches to Virtual Networking



OVS way



eBPF way

XDP



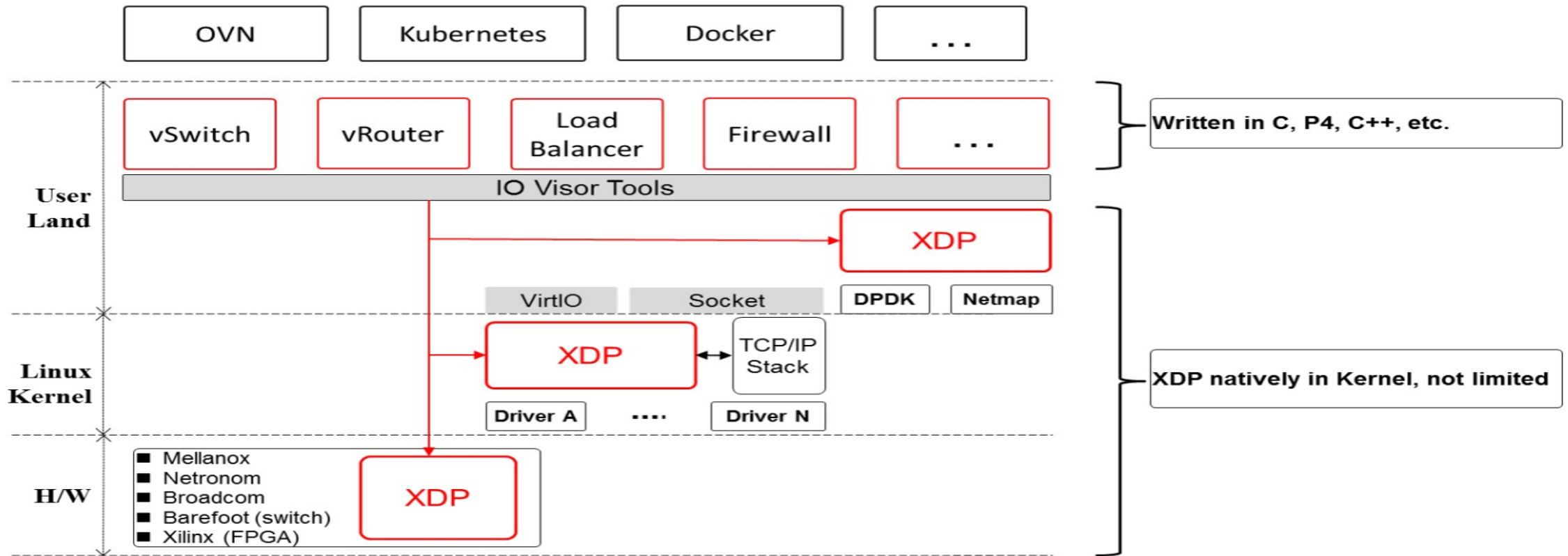
What is XDP?

- A programmable, high performance, specialized application
 - packet processor in the networking data path
- Hooks in supported drivers to attach eBPF programs
- Intercepts packets before they reach the stack
 - for more complex use cases send to stack
- Use cases include
 - vSwitch
 - vRouter (e.g. Facebook ILA Router)
 - Load Balancer
 - Security (Filtering, DDoS Mitigation)
- Linux 4.8+; Still under development

XDP Properties

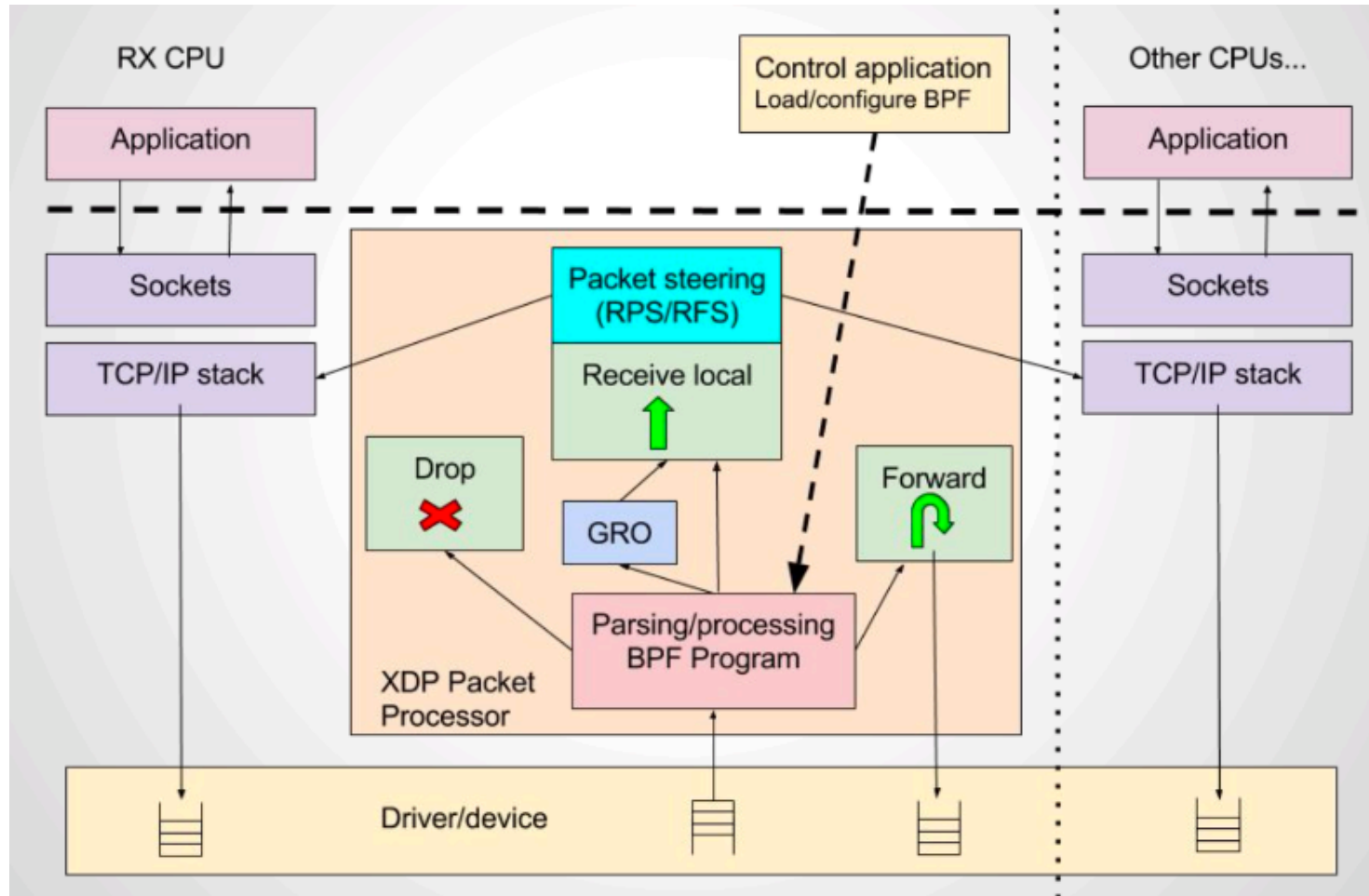
- XDP is **designed for high performance**. It uses known techniques and applies selective constraints to achieve performance goals
- XDP is also **designed for programmability**. New functionality can be implemented on the fly without needing kernel modification
- XDP is **not kernel bypass**. It is an integrated fast path in the kernel stack
- XDP **does not replace the TCP/IP stack**. It augments the stack and works in concert
- XDP **does not require any specialized hardware**. Less-is-more principle for networking hardware

XDP: Express Networking Data Path atop IO Visor



- IO Visor is the fundamental Framework/Platform
- XDP built for Express Network Data Plane

XDP Packet Processor



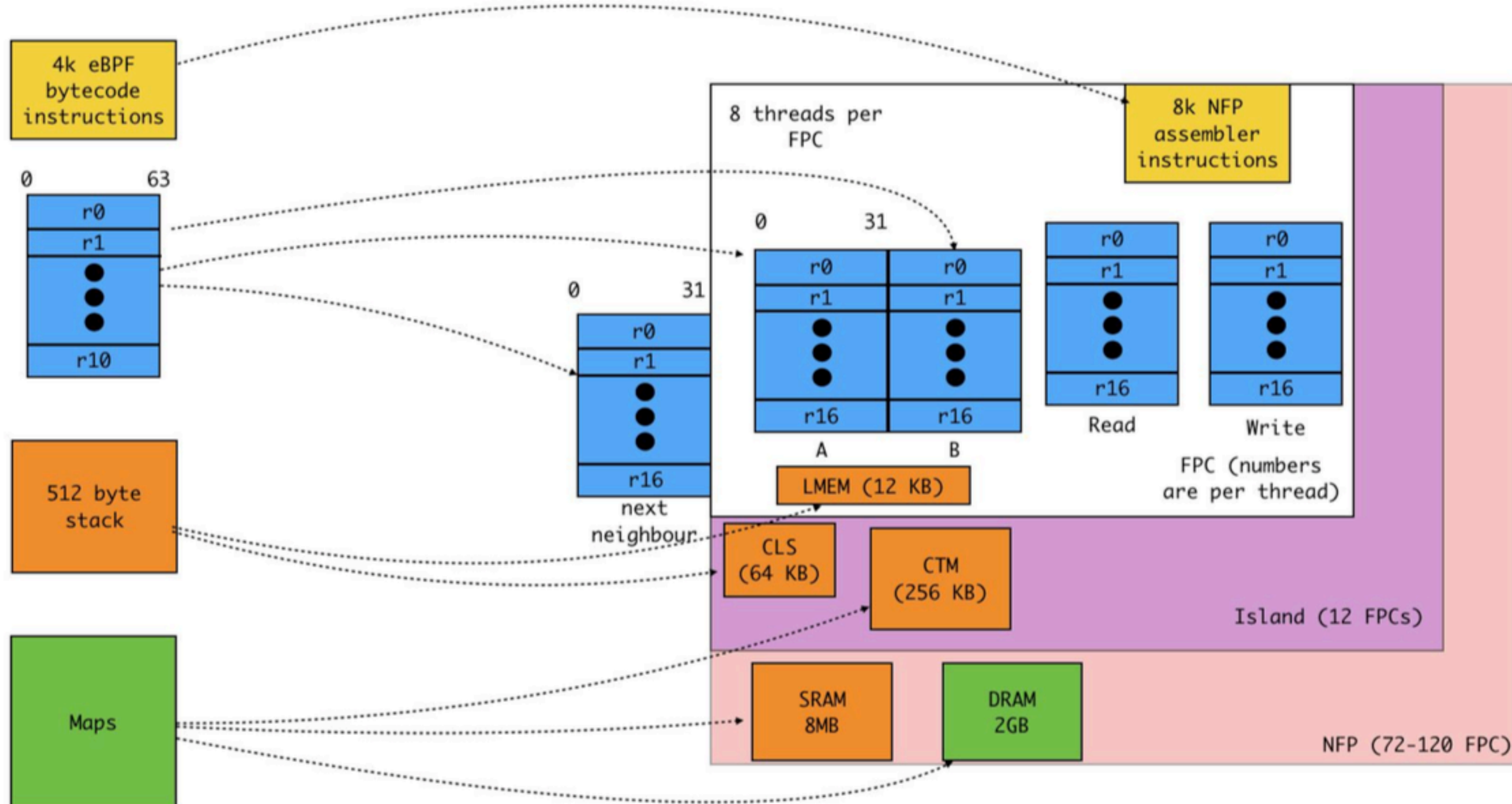
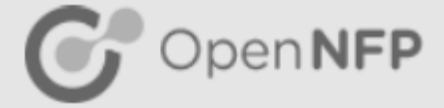
Accelerating eBPF in Hardware

The background of the slide features a large white triangle on the left side. To its right, the background is composed of several overlapping triangles in shades of green and blue. A light green triangle is at the top right, overlapping a darker green triangle. Below these, a light blue triangle overlaps a medium blue triangle, which in turn overlaps a dark blue triangle at the bottom right.

Accelerating eBPF in Hardware: why ?

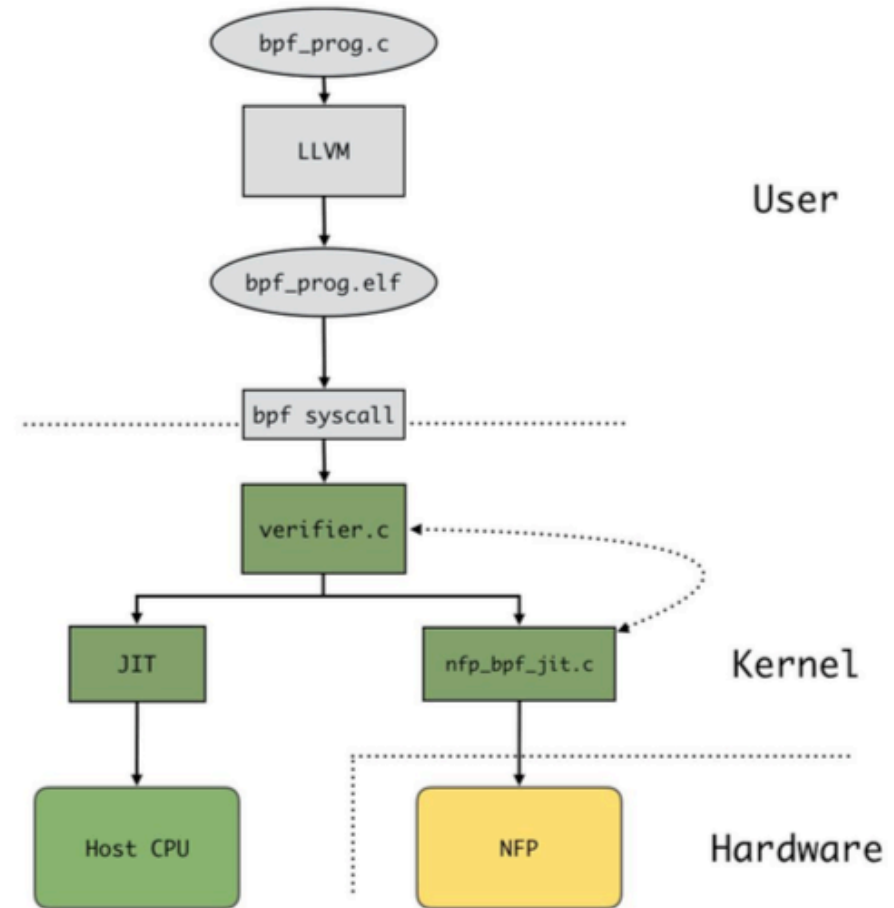
- eBPF is a well-defined framework
 - Parameters are well constrained registers, memory use, instruction set, helpers, etc...
- Implementation is ideal for multi-core architectures
 - Good for transparent offload to NPUs with many cores
- XDP and TC are the natural programming methods...
 - ...so the most likely targets for this type of generic networking offload

Mapping eBPF to Network Flow Processor (NFP)



Programming Model

- Program is written in standard manner
- LLVM Compiled as normal
- Then the *nfp_bpf_jit.c* converts the eBPF bytecode to NFP machine code
- Translation reuses a significant amount of verifier infrastructure
- This has motivated recent actions such as the creation of *bpf_verifier.h*



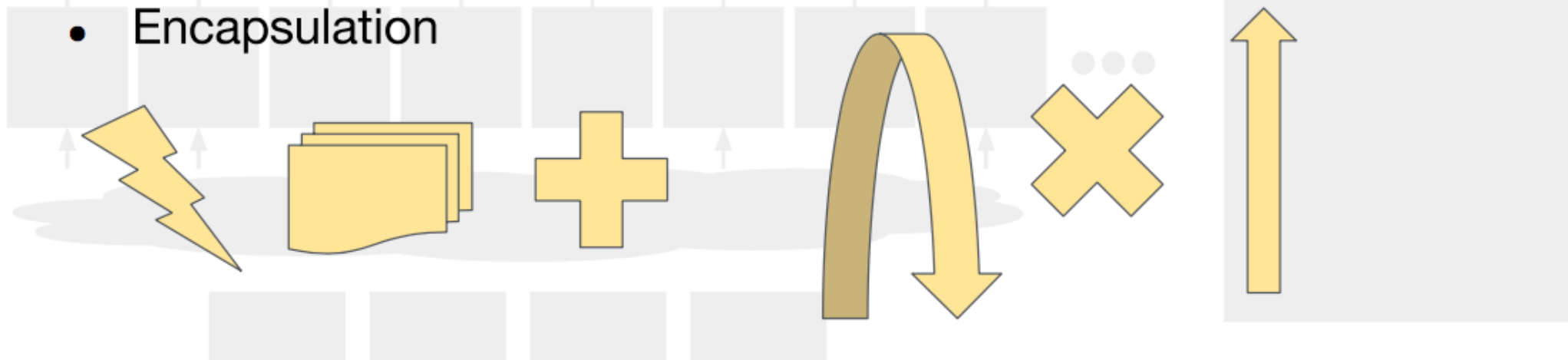
Operations and Actions

- ALU instructions except multiply and divide
- Packet Modification (header or payload)
- Basic maps
- Operations on relevant packet metadata fields
- Encapsulation

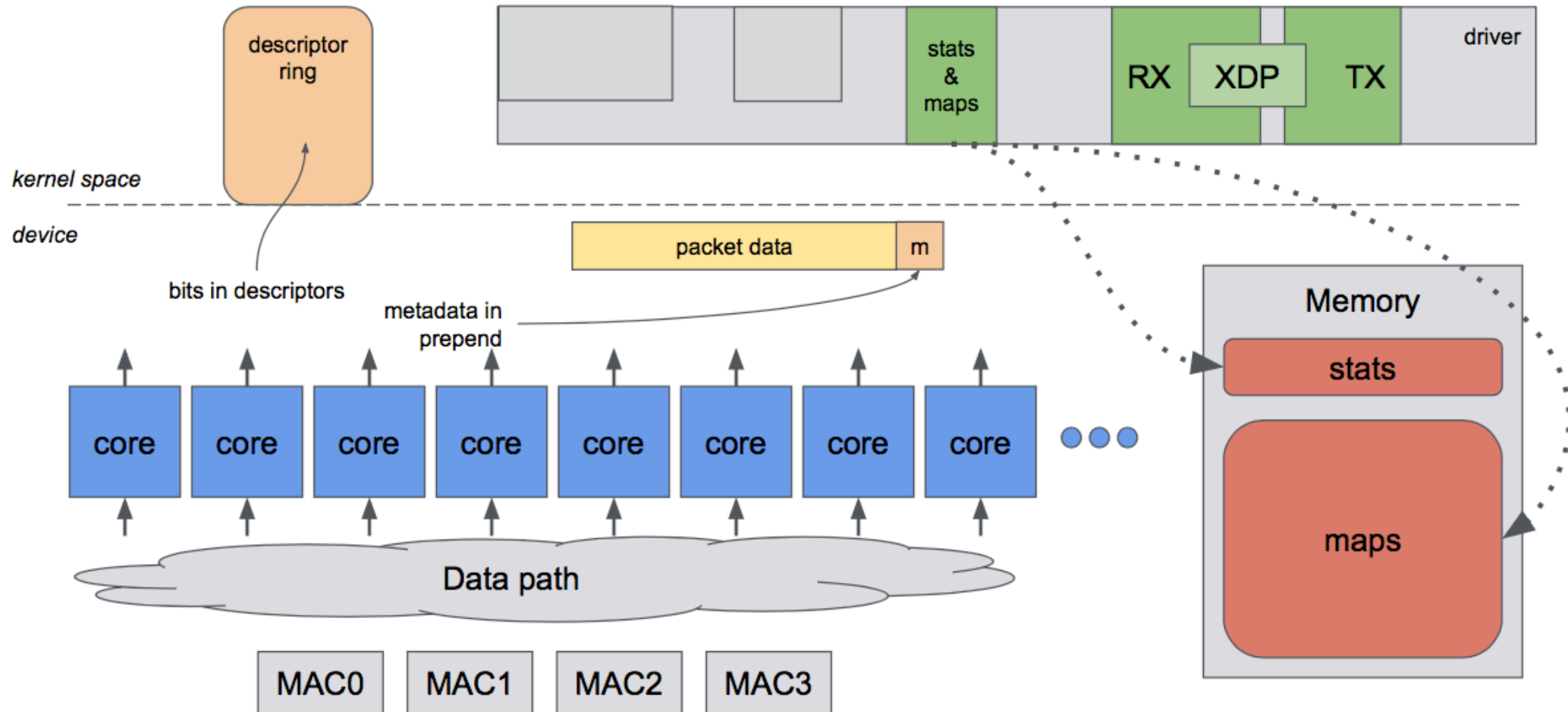
- Redirection
- Drop
- Pass (with modification and metadata)

kernel space

device



NFP details



Q & A

Thank You!



@cloudnativeapps
#vmwcna

vmware.github.io

blogs.vmware.com/cloudnative

microservices@vmware.com